UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2020

Title: Online Network Traffic Classification in Community Networks

Author: Jonathan Tooke

Project Abbreviation: DLOGs

Supervisor(s): Josiah Chavula

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	
Theoretical Analysis	*0*	*25*	5
Experiment Design and Execution	*0*	*20*	20
System Development and Implementation	*0*	*20*	10
Results, Findings and Conclusions	*10*	*20*	15
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	

Online Network Traffic Classification in Community Networks

Jonathan Tooke
jonathantooke@gmail.com
University of Cape Town
Cape Town, South Africa

ABSTRACT

Networks that support Quality of Service (QoS) configuration require an online network traffic classification tool to separate individual packets into their respective application classes at the network gateway. Machine learning techniques have proven to be effective for this classification task, even when the traffic is encrypted or routed through a VPN. Out of the machine learning techniques considered, deep learning models have typically attained the highest accuracy in research on traffic classification. However, research has not extensively considered the performance requirements of the models trained, incentivizing the development of larger, slower models in pursuit of higher accuracy. This poses a problem for online classification, particularly in low resource environments, where classification must be near instantaneous to avoid introducing unnecessary latency. This paper considers the trade-off between prediction speed and accuracy for the packet-based network traffic classification task by developing a framework that builds and compares hundreds of 1D CNN and MLP deep learning models of various sizes with varying payload lengths used as input. These deep learning models are further compared to an SVM across the same metrics. The models are evaluated using six different sets of hardware constraints that might be found in a modern community network, and the model that achieves the highest accuracy is selected for each of these resource environments, subject to it classifying the traffic efficiently enough. The study finds a clear trade-off between prediction rate and attainable accuracy. In this regard, it is only for resource environments with an exceptionally slow CPU that an SVM should be used. For low to middle-range CPUs an MLP can be used, and for the most powerful of CPUs, a 1D CNN is the preferred model.

CCS CONCEPTS

• **Networks** → **Network management**; • **Computing methodologies** → **Supervised learning by classification**; **Neural networks**.

KEYWORDS

deep learning, neural networks, network classification, community networks, quality of service, machine learning

1 INTRODUCTION

Network traffic classification is a problem that has undergone several evolutionary steps in line with improvements in network security standards and the growth of computational power. Improvements in network security, such as the reduction in dedicated application ports and an increase in the prevalence of encryption have made traditional network traffic classification algorithms less effective at best, and obsolete at worst [13]. At the same time, the growth in computational power and big data has resulted in the ever-increasing popularity of machine learning (ML) frameworks for this classification task.

The focus of this study is on packet-based online network traffic classification, which requires that traffic is classified using individual packets in near real-time. The online classification task will be considered from the perspective of a community network wanting to make use of QoS engineering, which requires a traffic classifier. Community networks typically experience slow internet speeds compared to their urban counterparts and rely on inexpensive hardware at the network gateway [4]. The slow internet speeds experienced in these networks make QoS engineering especially appealing as the benefits of prioritizing latency-sensitive packets becomes more pronounced when there is a bottleneck in the network. The resource constraints imposed by the inexpensive hardware in community networks means that a traffic classification tool must be lightweight and efficient enough to avoid adding latency to the network.

Deep learning techniques have shown significant success for the network classification task in recent studies [2, 8, 13]). Despite their capability of achieving higher accuracy on modern internet traffic than traditional ML [13], large deep learning models come with considerable computational overheads as compared to their traditional ML counterparts due to the sheer number of calculations that need to be performed. In low-resource environments, such as in community networks, it is not yet clear if deep learning models that are constrained in complexity by the network's computational limits would be capable of outperforming traditional ML models that are not as computationally intensive.

To investigate the trade-off between speed and model complexity, two deep learning models used in the network traffic classification literature are selected, the 1D Convolutional Neural Network (1D CNN) and the Multilayer Perceptron (MLP). Varying sizes of these models are built using a randomised grid search technique and their prediction speed and classification accuracy are recorded using a dataset from a community network. These models are further compared across the same metrics to a traditional and more lightweight ML model, the Support Vector Machine (SVM). Additionally, these models will be built with four different input lengths to investigate the potential performance gains of only considering a smaller section of each packet's payload as features.

To contextualise the performance of the models, six sets of community network hardware specifications are considered and the classification speed of the models on these hardware systems is compared to the classification speed required by the network to provide a model recommendation for each hardware specification from the set of all models trained. This model recommendation will be the model that achieves the highest accuracy subject to it being fast enough to run on the hardware system for online classification.

On the basis of related literature, the hypothesis is that the 1D CNN will outperform the MLP on accuracy [2]. Furthermore, the literature suggests that deep learning models performs better than traditional ML on modern network traffic, suggesting that the MLP and 1D CNN will outperform the SVM on accuracy [13]. It is likely that the SVM will be faster than the MLP which should be faster than the 1D CNN due to the comparative model complexities. The ability of each model to run efficiently enough on a community network low-resource environment is as of yet unknown. The effect on speed and accuracy of reducing the length of the payload as the input to the model is also unknown.

The rest of the paper is structured as follows. In section 2, some background is provided on QoS, community networks, and the ML models used in this paper. Section 3 provides a summary of the relevant related literature. Section 4 outlines the methodology followed by the paper. Section 5 provides the results and a discussion. Conclusions are given in section 6, and limitations, future work, ethical issues, and acknowledgements are provided thereafter.

2 BACKGROUND

In this section, some background is given on QoS and community networks to justify the need for a computationally efficient online network traffic classifiers. Thereafter, the theory behind the machine learning models used in this paper is discussed to give some insight into their inner-workings in the context of the network traffic classification problem.

2.1 Quality of Service (QoS)

QoS engineering is used to manage network traffic with the goal of reducing latency, packet loss and jitter. This can be done by prioritizing traffic from some applications over others according to predefined network rules [3]. Successful QoS engineering will improve user experience by ensuring that latency-sensitive traffic (such as video calls) can be prioritised over traffic that can be delayed for longer without frustrating users (such as email) . The router can only choose which packets to prioritize if it can first classify the incoming packets. This is done using online network traffic classification tools, such as the machine learning ones implemented in this paper.

2.2 Community Networks

Community networks are a solution to providing internet access in rural areas across the globe. They are typically formed by a small group of people coming together to develop a network infrastructure in their local community and then creating an access point to connect their network to the wider internet [9]. Community networks typically experience slow internet speeds, making QoS engineering particularly useful for ensuring a positive network experience. Furthermore, most community networks are built with inexpensive hardware [4], meaning that computationally intensive traffic classifiers, such as large deep learning models, may be too slow for online classification. Smaller deep learning models and traditional ML models may be required under these circumstances. The machine learning models trained in this paper use network traffic data sampled from the Ocean View community network in South Africa, meaning that the data is representative of network traffic coming from a community network.

2.3 Support Vector Machine (SVM)

The SVM is a traditional machine learning model that is typically used for binary classification problems. In a binary classification problem, the SVM attempts to distinguish classes along a maximum-margin hyperplane, which is essentially a straight line that splits the two data classes most convincingly in the features space [10]. In situations where the data is not linearly separable, a kernel function is used to transform the data into higher dimensions to allow for a linear separation. To prevent overfitting with this approach, and since most real-world datasets contain outliers, the basic SVM algorithm has been modified to allow for some values to fall on the wrong side of the hyperplane. This is known as the soft margin. In the case of multi-class classification, a one-versus-rest approach is used, where each class is separated from all of the other classes in the dataset. The one-versus-rest is the approach that is used in this paper.

2.4 Multilayer Perceptron (MLP)

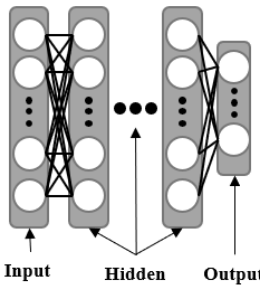


Figure 1: MLP Architecture

The multilayer perceptron is a type of feedforward artificial neural network. It consists of an input layer, one or more hidden layers, and an output layer. Each layer (l) consists of one or more nodes that are each connected to every node in the subsequent layer [12]. For each connection in the network, a weight value (w) is stored, and these values are adjusted as the network is trained. Additionally, each non-input node is assigned a non-linear activation function (σ) and stores a bias value (b) that is also learned during training. The data for the network is passed to the input layer, with each node in the input layer representing a feature of the data. Each subsequent node in the network calculates its output (a) by applying an activation function to the sum of the outputs of the previous layer multiplied by the connected weight and adding its bias value as shown by the equation below.

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \tag{1}$$

An MLP can be used for both regression and multi-class classification. In the case of multi-class classification, as used in this paper, the number of output nodes is set to be equal to the number

of classes to predict, where each output node is associated with one of the classes. A softmax activation function is used on the output layer, which results in each output node providing a predicted probability of the input data coming from its associated class.

Training is performed with the input data being divided into batches. After each batch passes through the network, the network outputs are compared to the true outputs using a loss function. An optimizer is used to reduce the error on the batch predictions by differentiating the loss function and using backpropagation (with some modifications depending on the optimizer) to adjust the weights and bias values in the network. Commonly used optimizers include stochastic gradient descent (SGD), RMSProp, and Adam optimization which is used in this paper [7]. The exact details of how these optimizers work is beyond the scope of this paper.

A technique known as regularization is commonly employed to reduce overfitting in these models. There are various regularization options available. In this paper, dropout is used, which randomly prevents some nodes in each layer from firing on each training iteration [14]. Another technique used is early stopping which stops training when the validation error starts to increase. Early stopping is also used in this paper for its regularization and training time advantages.

2.5 1D Convolutional Neural Networks (1D CNN)

A convolutional neural network is a more advanced deep learning model. The main difference to the MLP is that the input data is fed through convolutional layers first which performs the feature extraction [8]. Thereafter, a dimensionality reduction technique, such as max pooling is used, and the output of this is connected to one or more fully-connected layers.

In a 1D CNN, the convolutional layers take a one-dimensional vector as input. A kernel of some length is slid across each sub-region of the input data, calculating the dot product between the kernel and the sub-region of the input data at each point and appending the result to the output vector. This operation results in dependencies between subsequent features being captured by the model.

The equation for the output, z, from one kernel operation is given below. Where w is the kernel vector with a different weight value at each position a. The value l is the layer in the network and m is the length of the kernel.

$$z_i^l = \sigma(\sum_{a=0}^{m-1} w_a z_{(i+a)}^{l-1}) \tag{2}$$

Max pooling is typically used after convolutional layers in order to reduce the dimensions of the output. This works by taking the maximum value from subsequent sub-regions of the output and discarding the other values. The size of the sub-region can be chosen and tuned.

A similar approach for regularization can be used for CNNs as discussed in section 2.4 for the MLP, although dropout is less common following directly from convolutional layers. The same optimizer and output layer structure can be used as well.

3 RELATED WORK

Many different approaches to network traffic classification have been studied in related literature. Deep learning techniques have been studied in recent years in particular, partly due to the additional challenges imposed by the encryption of traffic [13]. Many of these studies perform flow-based classification based on inter-packet features. This is less suited to QoS which needs to perform the classification in real-time. As such, the key works that are particularly relevant to this study are those that use deep learning techniques for packet-based classification where network packets are classified individually. This approach is less commonly studied in the literature, meaning that there are not many papers to draw insights from.

Two main studies made use of 1D CNNs for encrypted traffic classification. The first used flow-based features [15], making it less relevant for online classification. The second used the packet payload as features, where each byte is a feature [8]. The second paper suggested that 1D-CNNs are ideal architectures for the traffic classification task using the packet payload since they are able to recognize dependencies between successive bytes in order to learn key patterns that enable successful classification. The paper explains that their deep learning models are able to learn the distinguishable patterns within the encrypted data that characterize applications, despite the content itself being inaccessible due to encryption. Their results show the effectiveness of their approach, with their 1D-CNN model attaining an F1 score of 0.95, outperforming all prior similar works in literature that perform classification on the same public dataset. This shows that 1D CNNs are promising for this packet-based classification task.

While the MLP has been studied for the flow-based traffic classification task, it does not seem that any research has been done on its effectiveness for packet-based classification. This leaves little to share about relevant related work for this model.

Packet-based classification using a 1D CNN has been shown to be successful. However, it does not appear that there have been any studies that consider computational resource utilization constraints on deep learning models, a critical constraint for online classification. Additionally, the effectiveness of MLPs for this packet-based task is yet to be discovered.

4 METHODOLOGY

The methodology was designed to ensure that the experiment can be easily reproduced. This begins with preprocessing the raw traffic data into the format required by the models. Thereafter, the specific requirements for the systems that the models must be able to run on as well as their required throughput rate were determined. Finally, the approach for building and evaluating the different models through the use of a randomised grid search over possible model architectures and hyperparameters is provided.

4.1 Preprocessing

Before building the traffic classifier models, the data needs to be transformed into the required input format. The data is converted to a packet-payload representation where each feature is a byte from the payload. The IP and TCP headers are excluded to prevent the model from using features such as port numbers and IP addresses

as features since these may change with time and are therefore not necessarily representative. A bash script is written to automate the transformation process from raw pcap input to csv output, and the steps followed by the script are outlined in full detail below. The script makes it straightforward to replicate the exact preprocessing steps followed in this paper. The preprocessing took approximately 3 days to run on a standard AWS EC2 server instance for the 14.2GB of the data used.

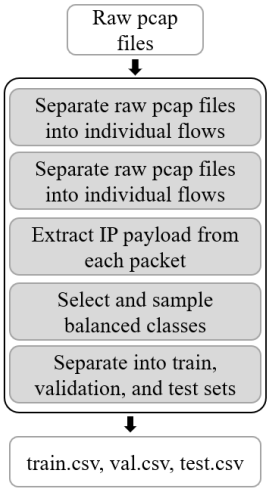


Figure 2: Preprocessing Steps

4.1.1 Dataset. The data used for training the models is collected from the network gateway of the Ocean View community network in South Africa [6]. A sample of 145 raw pcap files from this network, totalling 14.2GB in size, was used with the files picked over a date-range from March to May 2019 to ensure that the sample is representative over an extended time period.

4.1.2 Separate raw pcap files into individual flows. This step takes raw pcap files as input and produces the pcap flows extracted from the raw files as output. A flow consists of a number of packets that share the same source IP address and port, destination IP address and port, and protocol [13]. Each raw pcap file in the dataset consists of many flows that represent traffic from different application classes. The tool *pkt2flow*[1] was used to separate the flows in each raw pcap file and save each flow to its own pcap file.

4.1.3 Label flows with nDPI. This step takes pcap flows as input and produces a csv file with a label for each flow as output. The open-source deep packet inspection tool, *nDPI*[2], was used to produce the labels for each of the flows extracted in the step above.

4.1.4 Extract IP payload from each packet. This step takes the flows produced by pkt2flow and extracts the raw 1480 byte IP payload from each packet in each flow and saves them to a csv file where each byte is a feature as done in related literature [8]. For packets that have less than 1460 bytes in the payload, the remainder of the payload is zero-padded. Additionally, the label produced by nDPI is attached to this csv file, so that each packet payload has a label. The python library *scapy*[3] is used to perform the IP payload extraction.

4.1.5 Select and sample balanced classes. nDPI produced labels for 65 different classes, but only 10 of the classes were selected for the study. These classes were manually selected from the classes that had sufficient data. 10 000 packets were sampled from each of the 10 application class to ensure a balanced dataset, some of which originally had as much as 100 000 packets. Related literature suggested that this was a good amount of data per class [8] and using more data would have taken longer to preprocess and resulted in increased training time for the models. The classes selected can be found in the appendix.

4.1.6 Separate into train, validation, and test sets. The entire processed dataset consisting of 100 000 labelled byte-vectors (10 000 per class) is then split into train, validation, and test sets. 20% of the data is reserved for the test set, 16% for the validation set, and 64% for the training set.

4.1.7 Post-processing. One additional step was followed after the preprocessing was completed. The first 20 bytes of each packet were removed as they contained transport-layer header information which would allow the models to classify using port numbers. This is undesirable as the port numbers could change depending on the configuration of the application and system that it is running on.

4.2 Hardware Constraints Considered

The performance of the various models built in the experiment will be compared to the six resource constraints provided in the table below. The justification for considering these specifications can be found in the appendix.

Table 1: Resource constraints considered

	Effective CPU Speed
1	300 MHz
2	600 MHz
3	1.2 GHz
4	2.4GHz
5	4.8GHz
6	9.6GHz

A number of assumptions are made around the resource constraints. Firstly, there will always be a minimum of 25% of the CPU's resources available for the model to perform its classification. Secondly, differences in CPU architecture beyond clock speed are ignored and the models are assumed to be perfectly parallelizable where the effective CPU speed listed above is the number of CPU cores multiplied by the clock speed per core. Thirdly, the community network servers do not have access to a GPU. Fourthly, the RAM constraint is not considered. Lastly, the upload speed is fixed at 10Mbps which is used to calculate the required throughput.

[1] *pkt2flow* Available at: https://github.com/caesar0301/pkt2flow
[2] *nDPI* Available at: https://github.com/ntop/nDPI
[3] *scapy* Available at: https://github.com/secdev/scapy

4.3 Required Classification Throughput

The required throughput refers to the number of packets that need to be classified per second to match the number of packets transferred per second by the router. To calculate required throughput, the maximum network upload speed is divided by the expected size of each packet, where required throughput is the number of packets that must be classified per second. A network speed of 10Mbps (1250000 bytes/s) is based on the Ocean View community network. The expected packet size is calculated by taking the mean of the payload lengths of the sample taken from the Ocean View network, giving 970 bytes per packet. To be conservative, this value will be used to calculate throughput instead of including the IP header size as well. This produces the equation below.

$$RequiredThroughput = 1250000/970 = 1289 packets/s \quad (3)$$

4.4 Input Length

To investigate the effect of including less features in the input layer on model accuracy and performance, the packet lengths presented below are considered. When models are trained they will only take

Table 2: Number of bytes considered

Number of bytes (n)	Payload length
1460	100%
1095	75%
730	50%
365	25%

the first n bytes from the payload as input. The number of bytes will be varied across the different models built as per Table 2.

4.5 Evaluation Criteria

Classification accuracy is the metric chosen to evaluate the correctness of the models' predictions. Related literature frequently uses an F1 score, but this does not seem necessary as the classes used in this paper are perfectly balanced and as such there is no meaningful difference between false negatives and positives and true negatives and positives.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \times 100 \quad (4)$$

Additionally, the models are evaluated on the speed at which they can predict the test set of 20 000 packets on a CPU with a batch size of 32 and later compared to see if the prediction is sufficiently fast given varying resource constraints.

4.6 Architecture and Hyperparameter Selection

In order to find the best set of hyperparameters and model architectures for the varying resource constraints, a randomised grid search technique is used to search over both the model architecture and training parameters at the same time. The architecture parameters considered are designed to produce models of varying size and complexity.

4.6.1 Model architectures. MLP models are randomly constructed with between one and 3 hidden layers. 10 different values for the number of nodes in each layer are considered. However, the number of nodes considered is capped at the input size, reducing the number of options and complexity for smaller inputs used. For the CNN models, one or two 1D convolutional layers are used, followed by a max pooling layer, and then one or two dense layers. Details regarding the architecture parameters considered can be found in the appendix.

4.6.2 Hyperparameters. Other parameters searched over include the dropout rate, the batch size, the learning rate and the activation function. The input length (number of features) is also varied, but not at random. For the convolutional layers, values for filter size, kernel size, stride and padding are searched over. Different values for the max pooling layer are also considered. Details regarding the hyperparameters considered can be found in the appendix.

4.7 Experiment Procedure

4.7.1 Varying the input length. A method is written to transform the input data to the input lengths specified in section 4.4 so that the models can be built and evaluated on inputs of varying sizes.

4.7.2 Establishing a baseline with SVM. The first step of the experiment involves setting a baseline classification accuracy on the dataset with a traditional machine learning model using the scikit learn LinearSVC library (SVM classifier) [11]. The model was trained and evaluated across all 4 input lengths specified. A linear kernel along with a one-vs-the-rest scheme is used as per the scikit-learn documentation.

4.7.3 Training deep learning models. 300 MLP and 1D CNN models are trained across the hyperparamater and model architecture search space for each of the 4 input lengths outlined in section 4.4 using Keras [5] with Tensorflow [1]. This results in 1200 different MLP models and 1200 different CNN models that can be evaluated. The hyperparameters chosen for each of these models are saved to a csv file and the model itself is also saved. Early stopping is used to reduce training time and provide a regularization effect. A Tela V100 GPU is used to do this training and it takes between 24 and 48 hours to train each set of 1200 models.

4.7.4 Performance evaluation. The models are now loaded into a CPU instance where their prediction speed on the test set of 20 000 packets is recorded using a batch size of 32. The prediction speed is added to the csv file generated above, along with the size of the saved model in megabytes. The hardware specs for the CPU instance used is retrieved with a bash script and saved to a text file for later consideration.

4.7.5 Comparison to hardware constraints. A python script is written to take a csv file containing the model prediction speeds and the effective processing power of the system that produced that prediction speed. The ratio between the clock speed on the system that the model was tested on and the hardware constraints under consideration is calculated and the prediction rate is scaled by that same ratio. Thereafter, the calculated prediction rate is reduced by 75% to account for the assumption that only 25% of the system's

resources should be allocated to the traffic classification. The prediction rate is now compared to the required throughput calculated in section 4.3 to see if the hardware under consideration can support the model. An example showing how this is calculated is included in the appendix for additional clarity.

5 RESULTS AND DISCUSSION

In this section, the results of the experiment are reported. Insights are provided into the prediction rate and accuracy attainable for the different models as well as the trade-off between the two. Additionally, insights into the effect of reducing the payload length are given. For all of the graphs plotted below, prediction rate is calculated by predicting on the 20 000 packets in the test set and then calculating the number of packets predicted per second. This was performed on an Intel Xeon CPU with 2 cores with a clock speed of 2.30GHz per core (effective CPU speed of 4.6GHz). This rate is later adjusted to match the CPU requirements of the hardware constraints specified in the methodology section in order to provide a model allocation per hardware specification.

5.1 SVM Accuracy and Prediction Rate

Table 3: SVM Results

No. bytes (n)	Accuracy (%)	Prediction Rate (packets/s)
1460	**64.59%**	**142116**
1095	47.95%	193205
730	48.09%	260783
365	48.24%	322039

Table 3 shows that the SVM is capable of predicting at an accuracy of just under 65% when the full payload is used as the input. The prediction rate improves as the number of bytes included decreases, showing the improvement in performance of considering less features. However, this comes at the cost of a decrease in accuracy of just over 15% as soon as the full payload of 1460 bytes is no longer considered. There does not seem to be a significant difference between the accuracy of the SVM models that use less than the full payload, suggesting that the model does not extract any additional useful information from bytes 365 to 1095.

5.2 Model Accuracy Comparison

Figure 3 shows the difference between the distribution of accuracy on the test set for each of the deep learning models using each of the input lengths considered. It is clear that models with the full payload are capable of achieving a higher accuracy than models that use less bytes as input, but there does not seem to be a significant difference between the accuracy attainable for input lengths of any other size. The higher accuracy achieved by the 1460-byte models could be attributed to either some very useful features in the later bytes in the payload or that the payload length is a useful feature that the models can only learn when they have the full payload.

Table 4 shows a comparison between the maximum accuracy attained by the models of each class as well as the input length used and the prediction rate. There is a clear difference in accuracy across the classes, with the 1D CNN outperforming the MLP, which

Figure 3: Accuracy Distribution by Model and Input Length

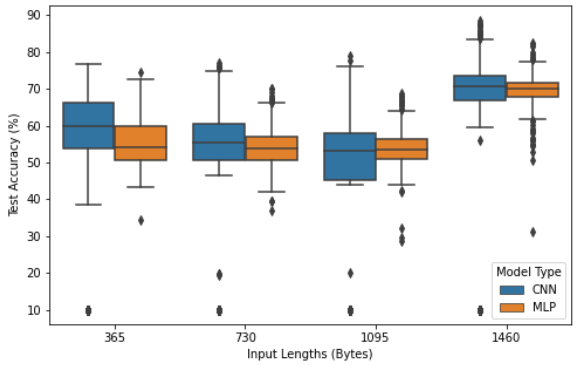


Table 4: Maximum Model Accuracy Comparisons

	1D CNN	MLP	SVM
Accuracy (%)	88.64%	82.48%	64.59%
Input Length (n)	1460	1460	1460
Prediction Rate (packets/s)	1115	34880	193205

outperforms the SVM. However, the prediction rate is about 30 times slower for the CNN as compared to the MLP, which is 6 times slower than the SVM. The hyperparameters used for these models can be found in the appendix.

5.3 Model Prediction Rate Comparison

Figure 4: Prediction Rate Distribution by Model and Input Length

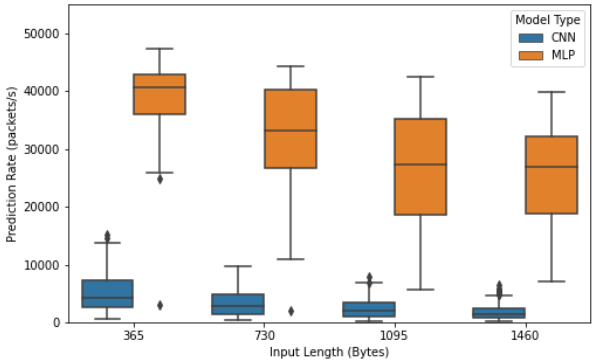


Figure 4 provides insight into the differences between the prediction rates of the MLP and the 1D CNN, as well as the differences between the prediction rates of models trained with the different

input lengths. It is clear from the diagram that the CNN is considerably slower than the MLP across input lengths of all sizes. Additionally, a trend of models performing faster with a smaller input length can be seen for both the MLP and CNN.

5.4 Maximum Accuracy Attainable by Input Length

Figure 5: Best Accuracy Attainable by Each Model for each Input Size

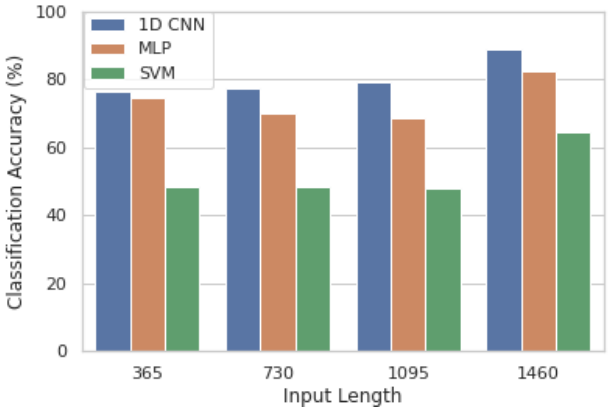


Figure 5 shows the maximum accuracy that each model class can attain for each input length. The highest accuracy attained by every model class was done with the full payload, indicating that the full payload should be used if accuracy is the only concern. However, section 5.3 showed that considering less features can result in a faster prediction rate, so it is possible that using a smaller payload could be beneficial for some hardware-model combinations. There does not seem to be a significant difference between the highest accuracy attainable for models with between 365 and 1095 bytes as input, suggesting that bytes 365 to 1095 do not add significant information.

5.5 Relationship Between Accuracy and Prediction Rate

Figure 6 shows the relationship between the test accuracy attainable and the prediction rate for the MLP models. There appears to be an inverse relationship, with the models attaining a higher accuracy at the cost of a slower prediction rate. The number of bytes considered by each model is also shown by adjusting the hue of the dots. A trend of the dots getting darker as the accuracy improves and then prediction rate declines can also be observed, indicating that the smaller input sizes demonstrate faster classification at the expense of accuracy for the MLP models.

Figure 7 shows the relationship between the test accuracy attainable and the prediction rate for the 1D CNN models. A similar inverse relationship exists between test accuracy and prediction rate, but it is worth noting that the models are considerably slower than those presented in Figure 6. As with Figure 6, a trend of the dots getting darker as the accuracy improves can be seen indicating that the smaller input sizes demonstrate faster classification at the expense of accuracy for the 1D CNN models.

Figure 6: MLP Accuracy vs Prediction Rate for Models with Accuracy above 65%

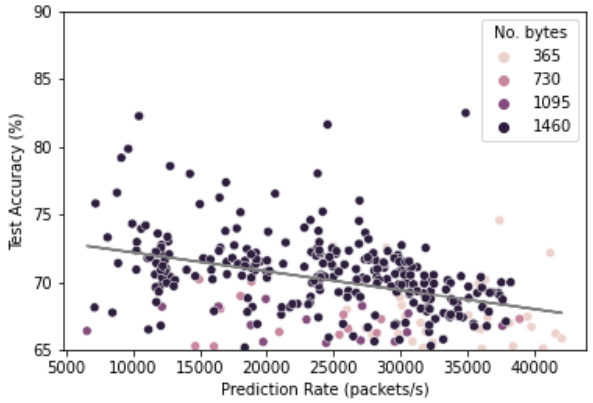
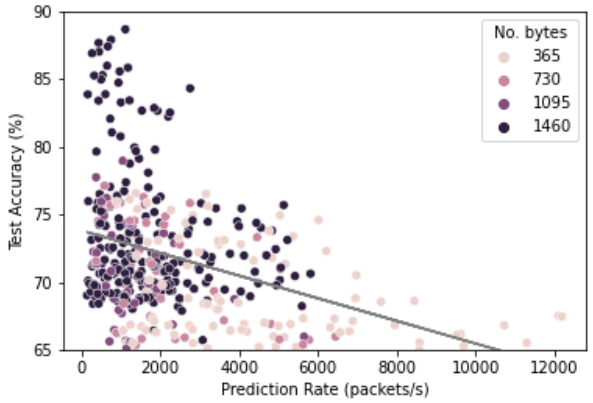


Figure 7: 1D CNN Accuracy vs Prediction Rate for Models with Accuracy above 65%



5.6 Model Allocations by Hardware Constraints

The following model allocations are made for the resource constraints outlined in the methodology section. Note that the CPU speed in the table is taken after taking the clock speed and reducing it to the 25% available for classification. Furthermore, the prediction rate is adjusted to be representative of what might be expected on each of the CPUs under consideration, and this adjusted rate is used to allocate the model with the highest accuracy that can meet the required throughput rate of 1289 packets/s for each of the CPUs.

The model name is a unique identifier which follows the format [Model class]_[Number of bytes used as input]_[id] where id is a number between 0 and 299.

Table 5: Model Allocations by CPU speed

CPU Available (GHz)	Model Name	Accuracy	Rate (pk/s)
0.075	SVM_1480	64.59%	2317
0.15	MLP_365_136	72.16%	1343
0.3	MLP_1460_127	82.48%	2274
0.6	MLP_1460_127	82.48%	4549
1.2	MLP_1460_127	82.48%	9099
2.4	CNN_1460_197	84.29%	1438

Table 5 shows which model was chosen for each hardware system, where the model with the highest classification accuracy is chosen subject to it meeting the required prediction speed on the given hardware system. Firstly, it is only the weakest processor that resorts to using an SVM with an accuracy of 64.59% and only the most powerful processor that selects a CNN with an accuracy of 84.29%. The others all use the same MLP model with an accuracy of 82.48%, aside from the second weakest processor which uses an MLP with 365 bytes as input and produces an accuracy of 72.16%. The highest accuracy CNN reported in section 5.4 is too slow even for the fastest CPU considered. The hyperparameters for these models can be found in the appendix.

6 CONCLUSIONS

A number of key conclusions can be drawn from the results of the experiment and they are outlined below.

6.1 Highest Accuracy Model Performances

Similar to related literature, the 1D CNN has performed better than the MLP on accuracy. However, it is significantly slower than the MLP, with the highest accuracy 1D CNN predicting 30 times slower than the highest accuracy MLP. The highest accuracy 1D CNN achieved an accuracy of 88.64% and the highest accuracy MLP achieved an accuracy of 82.48%. The best SVM model uses the full payload as features and achieves an accuracy of 64.59%, significantly worse than the CNN and MLP, but performs 6 times faster than the best MLP and 180 times faster than the best 1D CNN. The highest accuracy 1D CNN is too slow for even the fastest of the resource constraints considered.

6.2 Inverse Relationship between Prediction Rate and Test Accuracy

It has been shown that there is a general inverse relationship between the prediction rate and the attainable accuracy, both within model classes and between model classes. This means that low-resource environments may have to select a faster model with a lower accuracy to meet the throughput requirements of an online traffic classifier.

6.3 Full Payloads Achieve Higher Accuracy at the Cost of Performance

The analysis on considering payload lengths of less than the full 1460 bytes revealed that this does come at a significant accuracy cost, but with a faster prediction rate. Despite performing worse than full-payload models, the models with 365, 730, and 1095 bytes as features performed similarly to each other, suggesting that there is not meaningful information captured from bytes 365 to 1095. Despite the lower accuracy, considering smaller input lengths can be worthwhile lower resource environments as is shown by MLP_365_136 in Table 5 which only uses the first 365 bytes as input and was selected for the second slowest resource environment.

6.4 Community Network Recommendation

The recommended model for a given community network depends on their required throughput, available processor resources, and available RAM. Only the first two have been considered here, but it is reasonable to infer that models with a faster prediction rate will generally require less RAM as less calculations are performed and thus less values need to be stored.

For most community networks, it appears that the MLP is a good option for network classification as it seems to perform sufficiently fast to meet the processor constraints of most community networks while achieving a reasonable accuracy. The SVM should only be used in community networks that have exceptionally slow processors. The CNN should be used in community networks with high-end CPUs or access to a GPU.

The process of selecting a model can be easily replicated for a network with arbitrary resource constraints as a function has been written that will take the constraints specified and return the model that can attain the highest accuracy subject to the constraints.

For actual implementation in a community network with known resource constraints, this paper can serve as a guide as to what sort of models could be supported by the network and a model could be fine-tuned to achieve an accuracy marginally higher than the ones generated by the random search in this paper.

7 LIMITATIONS

Although interesting and suggestive, the results in this paper should not be considered conclusive. A number of key limitations must be kept in mind when interpreting the results, particularly since the deep learning field of work is a highly experimental one.

7.1 Limited Search Space

The search space for the architectures and hyperparameters of the 1D CNN and MLP models considered is infinitely large. Hyperparameters and architectures were selected based on what is commonly done in machine learning, but it by no means implies that there were not better hyperparameter and architecture combinations that may have attained a higher accuracy without performance degradation. As such, the results should only be considered as significant within the search space considered. Furthermore, a limit on access to computational resources meant that it took a lot of time to train and test models, which prevented the expansion of the search space.

7.2 Class Selection

The performance of the models may depend on the classes selected. If a community network were to implement a deep learning classifier they may wish to include additional or different applications for classification to the ones considered by the study. The effect of changing the number of classes considered on accuracy and prediction rate has not been investigated and neither has the effect of using different classes with a different distribution.

7.3 Handling Data from Unknown Classes

The accuracy of the models trained is not representative of what would be encountered in a live network, as the models assume that the 10 classes considered are the only classes in the network and would incorrectly classify traffic from an arbitrary class as traffic from one of the 10 classes. To test how the models handle such traffic, data representative of standard network traffic should be fed into the models and only situations where the model is sufficiently confident in its prediction, should it produce a label. Alternatively, the models could be trained with an 'other' class where it attempts to classify traffic as belonging to a different class as the ones under consideration.

7.4 Data Labelling

The labels used in the dataset were generated by nDPI, a deep packet inspection tool, as mentioned in the preprocessing stage. One of the issues with this approach is that the accuracy of the models on the underlying data is limited to the accuracy of the data labelled by nDPI, as the test set used also uses labels from nDPI.

8 FUTURE WORK

This study has provded valuable insights into the potential of using deep learning models for the purposes of online traffic classification in low resource environments. However, there are a number of ways in which the work could be extended upon.

8.1 Comparison with other ML Techniques

This study only considered the effectiveness of the 1D CNN, MLP and SVM on for the online traffic classification task. It is possible that there are other models that would provide a better combination between performance and accuracy than those considered. Other deep learning models used in related literature with good performance include the LSTM, 2D CNN and Stacked Autoencoder. The performance of other traditional machine learning models could also be investigated and compared to the SVM.

8.2 Varying the Number of Classes to Predict

It may be useful to investigate the effect of considering a different number of classes for classification on accuracy and performance. If a negative relationship is shown between the number of classes considered, and accuracy and performance, this would suggest that the number of classes considered should be kept to a bare minimum. Alternatively, if a negative relationship is not found, then this would allow network engineers to classify more types of traffic without a loss in performance or accuracy.

9 ETHICAL, PROFESSIONAL, AND LEGAL ISSUES

The project faces few ethical, professional, and legal issues. This is largely due to the fact that the team is not interacting with individual people, or sensitive data that can be traced to individuals. The most important ethical issue to consider is the confidentiality of the aggregated network data that has been provided by the Ocean View community network in South Africa for the purposes of training and testing our models. The network has granted permission for the team to use the data for research purposes. However, it is still important that the privacy of the network is protected, so the team will ensure that the data is only accessed by the team and not distributed publicly. There are no legal issues faced by the project.

Upon completion of the project, the source code will be made public for future researchers to use and build upon. If the research paper is published, it will be published under the creative commons license as per the University of Cape Town's policy.

10 ACKNOWLEDGEMENTS

Thanks to my supervisor, Josiah Chavula who has always been available to assist by offering guidance and answering questions despite the online nature of the year. A big thank you to my team members, Matthew Dicks and Shane Weisz for the group effort on the preprocessing and for being available to discuss ideas for the project at large. Lastly, thank you to the UCT Computer Science Department staff for their substantial role in my computer science education.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé. 2019. Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges. *IEEE Transactions on Network and Service Management* 16, 2 (2019), 445–458.

[3] A. O. Adedayo and B. Twala. 2017. QoS functionality in software defined network. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. 693–699.

[4] Bart Braem, Chris Blondia, Christoph Barz, Henning Rogge, Felix Freitag, Leandro Navarro, Joseph Bonicioli, Stavros Papathanasiou, Pau Escrich, Roger Baig Viñas, Aaron L. Kaplan, Axel Neumann, Ivan Vilata i Balaguer, Blaine Tatum, and Malcolm Matson. 2013. A Case for Research with and on Community Networks. 43, 3 (2013).

[5] Francois Chollet et al. 2015. *Keras.* https://github.com/fchollet/keras

[6] iNethi. 2020. *Community Network Deployments.* Retrieved 2020-09-19 from https://www.inethi.org.za/deployments/

[7] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[8] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. 2019. Deep packet: a novel approach for encrypted traffic classification using deep learning. *Soft Computing* 24, 3 (2019), 1999–2012. https://doi.org/10.1007/s00500-019-04030-2

[9] P. Micholia, M. Karaliopoulos, I. Koutsopoulos, L. Navarro, R. Baig Vias, D. Boucas, M. Michalis, and P. Antoniadis. 2018. Community Networks and Sustainability: A Survey of Perceptions, Practices, and Proposed Solutions. *IEEE Communications Surveys Tutorials* 20, 4 (2018), 3581–3606.

[10] William S Noble. 2006. What is a support vector machine? *Nature biotechnology* 24, 12 (2006), 1565–1567.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[12] Hassan Ramchoun, Mohammed Amine, Janati Idrissi, Youssef Ghanou, and Mohamed Ettaouil. 2016. Multilayer Perceptron: Architecture Optimization and Training. *International Journal of Interactive Multimedia and Artificial Intelligence* 4, 1 (2016), 26. https://doi.org/10.9781/ijimai.2016.415

[13] S. Rezaei and X. Liu. 2019. Deep Learning for Encrypted Traffic Classification: An Overview. *IEEE Communications Magazine* 57, 5 (2019), 76–81.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[15] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang. 2017. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 43–48.

A CLASSES SELECTED

The 10 application classes used in the study include the following:

- BitTorrent
- GoogleDocs
- Instagram
- GMail
- YouTube
- Cloudflare
- WhatsApp
- GoogleServices
- Facebook
- TeamViewer

B HARDWARE CONSTRAINTS

B.1 Calculation Example

In this section, an example is included to show how the prediction rate is scaled for the various hardware systems under consideration. Note, this calculation relies on a number of simplifying assumptions that are outlined and justified in the methodology section.

A model is trained on a system with 2 cores each operating at 2.3GHz. This is converted to an effective CPU speed of 4.6GHz by adding the speed of the two cores together. The model records a prediction rate of 10000 packets per second on this system (i.e. it classifies the full test set of 20000 packets in 2 seconds). The required throughput calculated in the methodology section is 1289 packets/s for a 10Mbps line.

The system under consideration is a community network with an effective CPU speed of 2.3GHz. To convert the classification rate to be representative of what could be done on this system, the ratio of the clock speeds is calculated (2.3/4.6 = 0.5) and the original prediction rate is multiplied by this number resulting in a prediction rate of 5000 packets/s (10000*0.5). This number is then reduced by 75% to account for the fact that only 25% of system resources should be used for classification, resulting in a throughput of 1250 packets/s. This is too slow for the community network under consideration and the model is rejected.

B.2 Constraints Considered

- 300 MHz single core ARM processor as found on very low-range home routers.
- 600 MHz single core ARM processor as found on low-range home routers.
- 1.2 GHz single core ARM processor as found on middle-range home routers.
- Intel Pentium 4 Processor. 1 core @ 2.80 GHz. 2GB RAM. As found in entry level servers.
- Intel Core i3-5010U process. 2 cores @ 2.10 GHz/core. As found in some community network servers.
- Intel Core i5-8259U processor. 4 cores @ 2.3Ghz per core. As found in some community network servers.

C ARCHITECTURES AND HYPERPARAMETERS CONSIDERED

For each model constructed an item is selected randomly from the parameters provided in the lists below.

C.1 MLP

- Number of layers: [1,2,3]
- Number of nodes per layer: [8, 16, 32, 64, 128, 256, 512, 1024], but does not select a number of nodes for each layer that is greater than the input length
- Activation function: [relu, selu, sigmoid, swish]
- Dropout from input layer: [0, 0, 0.05, 0.1, 0.15, 0.2]
- Dropout between hidden layers: [0, 0, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
- Dropout to the output layer: [0, 0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
- Learning rate: [0.002, 0.0015, 0.001, 0.001, 0.0009, 0.0008]

C.2 1D CNN

- Number of convolutional layers: [1,2]
- Number of filters: [20, 50, 75, 100, 125, 200]
- Kernel size: [2, 3, 4, 5, 6]
- Stride length: [1,2,3]
- Padding: [valid, same]
- Max pooling size: [2, 4, 6, 8]
- Activation function: [relu, selu, sigmoid, swish]
- Learning rate: [0.002, 0.0015, 0.001, 0.001, 0.0009, 0.0008]
- Number of hidden layers: [1,2]
- Number of hidden nodes: [30, 50, 80, 100, 120, 200, 300]
- Dropout to hidden nodes: [0, 0, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
- Dropout to output layer: [0, 0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]

C.3 Other

- Batch size: [64, 128, 256, 512, 1024]

D NOTABLE MODEL PARAMETERS

D.1 MLPs

MLP_1460_127 (highest accuracy MLP)

- Number of layers: 3
- Number of nodes per layer: [32, 32, 256]
- Activation function: relu
- Dropout from input layer: 0
- Dropout between hidden layers: 0
- Dropout to the output layer: 0.35
- Learning rate: 0.002
- Batch size: 128
- Number of bytes as input: 1460

MLP_365_136

- Number of layers: 3
- Number of nodes per layer: [16, 64, 32]
- Activation function: selu
- Dropout from input layer: 0.05
- Dropout between hidden layers: 0
- Dropout to the output layer: 0.1
- Learning rate: 0.0015
- Batch size: 64
- Number of bytes as input: 365

D.2 1D CNNs

D.2.1 CNN_1460_177 (highest accuracy CNN).

- Number of convolutional layers: 2
- Number of filters: [100, 20]
- Kernel size: [5, 6]
- Stride length: 2
- Padding: same
- Max pooling size: 6
- Activation function: relu
- Learning rate: 0.001
- Number of hidden layers: 1
- Number of hidden nodes: 50
- Dropout to hidden nodes: 0.3
- Dropout to output layer: 0.5
- Batch size: 64
- Number of bytes as input: 1460

D.2.2 CNN_1460_197.

- Number of convolutional layers: 2
- Number of filters: [20, 20]
- Kernel size: [5, 4]
- Stride length: 1
- Padding: same
- Max pooling size: 2
- Activation function: relu
- Learning rate: 0.002
- Number of hidden layers: 1
- Number of hidden nodes: 100
- Dropout to hidden nodes: 0.45
- Dropout to output layer: 0.1
- Batch size: 256
- Number of bytes as input: 1460